

Object-oriented Development and Testing

The CatShop Project

<https://github.com/aeh73/catshop> - Private

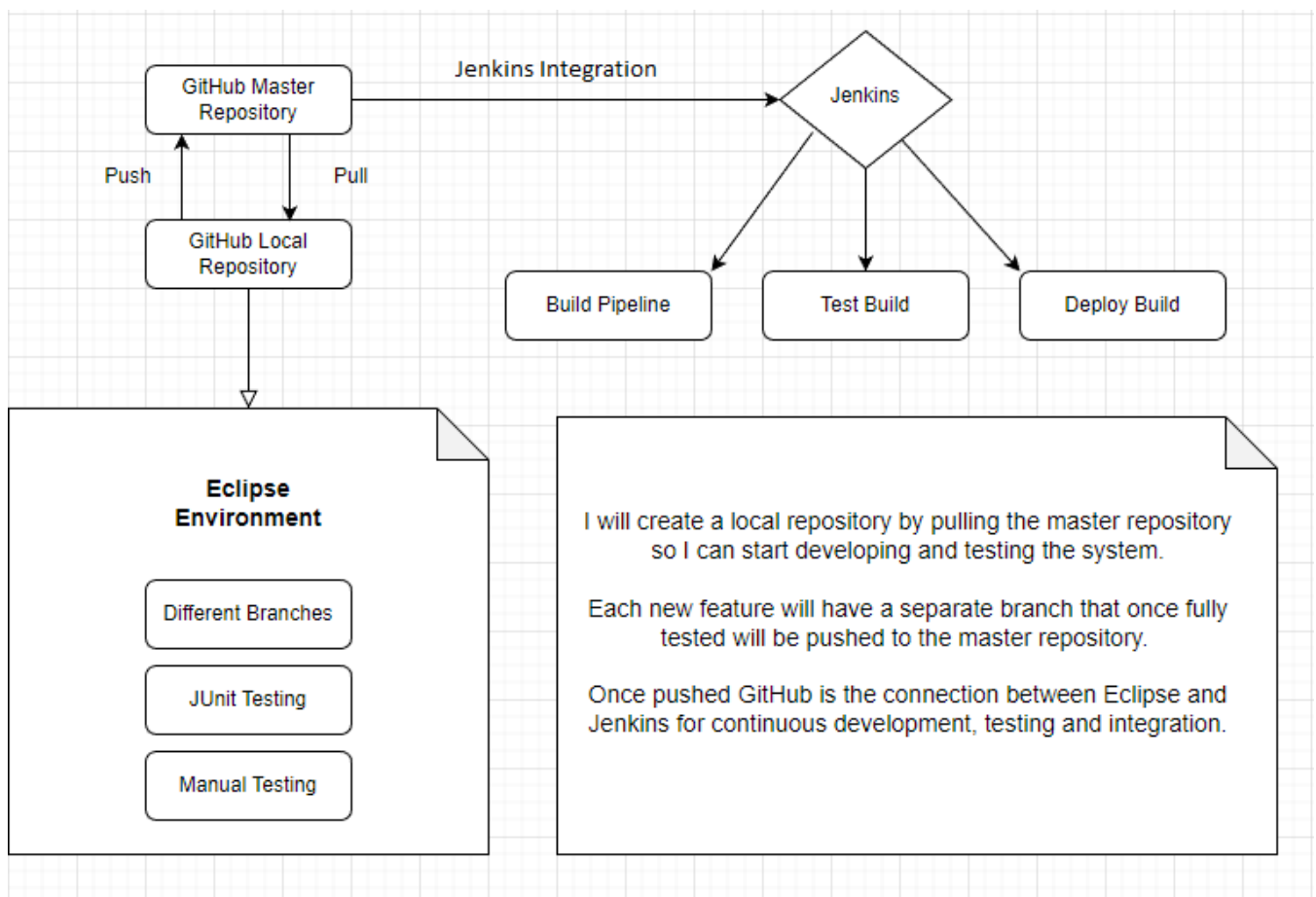
Introduction

After trying out the program there are a few improvements I'd like to introduce - firstly I'd like to improve GUI with background images, button colours and sounds, next I'd like to improve the Basket class to a BetterBasket class and BetterCashier class using the labs and also the functionality of the CashierModel and application in general.

I will be developing the system using a combination of GitHub and Jenkins for development and testing and UML to better visualise the classes and how they interact.

To do this I'll be creating a new branch on my local GitHub repository that can be later pushed to the master repository after regression/manual testing, JUnit testing and finally using Jenkins to automate building, testing and deploying the pipelines.

To better illustrate my project cycle.

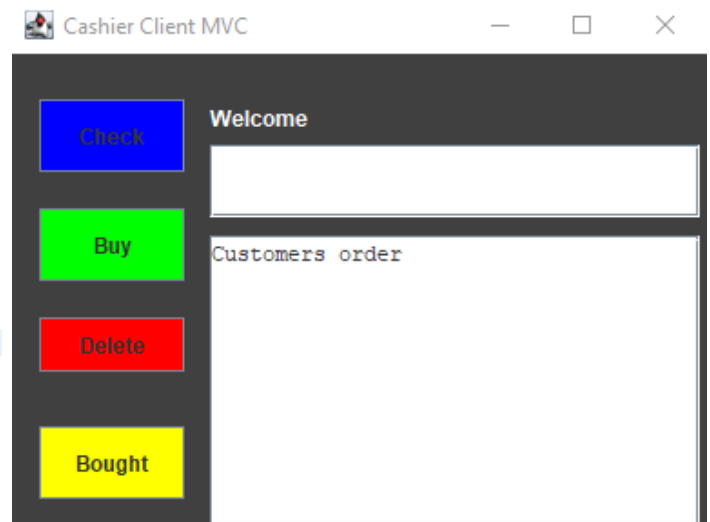


Development and Management

To begin with I wanted to improve the aesthetics of the GUI giving it a dark mode feel and incorporate some sounds, I cloned the coursework repository from GitHub and created a new branch 'upgradeGUI'.

To give it a dark mode feel I simply set the background colour of the content pane to 'DARKGRAY', similarly changing the colours of the buttons you used .setBackground(Color.[]) and then finally changed the text colour to something visible like white.

```
Container cp          = rpc.getContentPane();    // Content Pane
Container rootWindow = (Container) rpc;         // Root Window
cp.setBackground(Color.DARK_GRAY);
theBtCheck.setBackground(Color.BLUE);
theBtBuy.setBackground(Color.GREEN);
theBtBought.setBackground(Color.YELLOW);
theAction.setForeground(Color.WHITE);
```



To integrate sounds I needed to create a public sound class with a play method that took a string filename as a parameter that could be called upon throughout the application.

```
/*sounds class with a play method which will take in a filename as a string parameter*/
public class sounds {

    public void play(String fileName) {
        if(fileName!=null) { //aslong as the filename is not null
            File file= new File(fileName+".wav"); //then create a newfile with the filename and ".wav" as an extension
            try {
                AudioInputStream audio = AudioSystem.getAudioInputStream(file); //obtains an Audioinputstream of the user defined format
                Clip clip=AudioSystem.getClip(); //AudioSystem retrieves the audio file and assigns it to clip
                clip.open(audio); //Open the clip and start the audio
                clip.start();
            } catch (UnsupportedAudioFormatException | IOException | LineUnavailableException e) {
                System.out.println("ERROR");
                e.printStackTrace();
            }
        }
    }
}
```

(Rosencrantz, N. *playing sound in java*, 2020)

And for the actual sounds I wanted a start-up jingle when loading the application and different sounds for button click events.

```
public void begin()
{
    sounds sounds = new sounds();
    sounds.play("starup");

    public void doCheck( String pn )
    {
        model.doCheck(pn);
        sounds sounds = new sounds();
        sounds.play("chkSnd");
    }
}
```

On start-up in Main.java I created a new sounds object from my sounds class that used the play method with the filename passed as a parameter.

To have individual sounds for each button I needed to apply the same method in each classes' controller, the image is from my CashierController which had different sounds for each button.

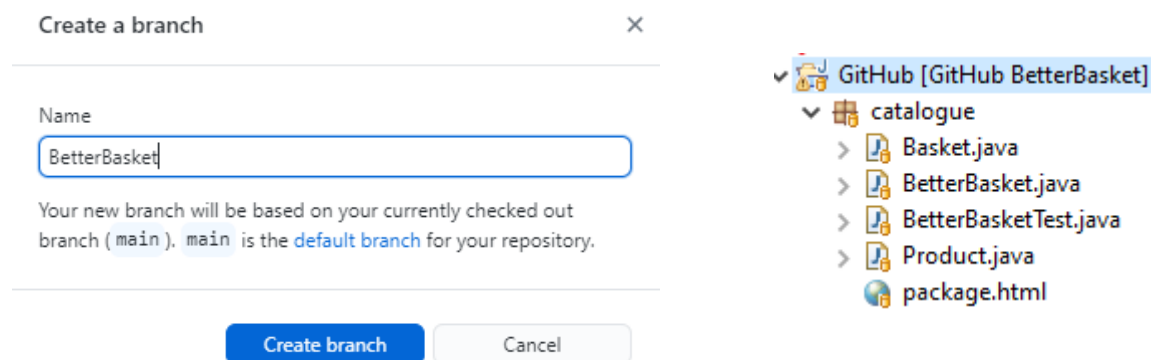
Testing for the sounds and GUI upgrades were done manually.

BetterBasket/BetterCashier

Next area to improve was the Basket class, originally it would add the same product separately rather than increasing its quantity as it simply uses the ArrayList method .add which will just add it to the end of the ArrayList.

To improve this functionality, I created a BetterBasket branch and created a BetterBasket class which extended the original basket class and created a BetterBasketTest class in preparation.

Using the GitHub desktop application, I can create branches and integrate it with eclipse with ease.



Once the branch is created, I can start modifying code and testing it, before publishing and pushing it.

```
public class BetterBasket extends Basket implements Serializable, Comparator<Product>
{
    private static final long serialVersionUID = 1L;

    @Override
    public boolean add(Product p1) {
        /*Improved code from 2019 lecture slides*/
        //loop through products to see if there is a match
        for (Product p2: this) {
            // checks if what we adding is already in basket
            if(p1.getProductNum().equals(p2.getProductNum())) {
                //if found update the quantity and return true
                p2.setQuantity(p2.getQuantity()+p1.getQuantity());
                return true;
            }
        }
        //else it was not found using the superclass method we add a new product
        super.add(p1);
        //To sort the list we turn BetterBasket into a comparator object
        //and give it a compare method to products
        Collections.sort(this, this);
        return true;
    }

    public int compare(Product p1, Product p2) {
        return p1.getProductNum().compareTo(p2.getProductNum());
    }
}
```

The BetterBasket class Inherits from the Basket class and overrides the add method with an improved version.

The new add method will loop through the ArrayList of products checking to see if the new product matches an existing product, if so, increment the quantity rather than adding a new product line – else the product is not in the basket so we can use the superclass method in adding the product.

Finally, we sort the ArrayList using Collections.sort this is done by turning the basket of products into a comparator object and giving it the compare method – a simple method which compares Product p1 to Product p2.

To implement the BetterBasket class we will need to alter 1 class and create another – the main.java and BetterCashierModel.java

```
CashierModel model = new BetterCashierModel(mf);
```

In the main.java we changed this line of code to retrieve the BetterCashierModel instead of the original CashierModel.

```
/*BetterCashierModel extends CashierModel and overrides the makeBasket
 * Method to return BetterBasket instead*/
public class BetterCashierModel extends CashierModel {

    public BetterCashierModel(MiddleFactory mf) {
        super(mf);
    }

    @Override
    protected Basket makeBasket() {
        return new BetterBasket();
    }
}
```

Next, we created the BetterCashierModel class which extended the original CashierModel and overrides the makeBasket method to return a new BetterBasket.

To test this functionality, I incorporated Junit testing, manual testing and included a Jenkins build, to do this we added a few tests to our BetterBasketTest class.

```
public class BetterBasketTest {

    @Test
    public void testMergeAddProduct() {
        BetterBasket b = new BetterBasket();
        Product p1 = new Product("0001", "Toaster", 10.00, 1);
        Product p2 = new Product("0001", "Toaster", 10.00, 1);
        Product p3 = new Product("0002", "Blender", 20.00, 1);
        Product p4 = new Product("0002", "Blender", 20.00, 2);

        //Test that p1 and p2 get merged
        b.add(p1);
        b.add(p2);
        assertEquals(1, b.size(), "Size Error");
        assertEquals(2, b.get(0).getQuantity(), "Quantity Error");

        //Test that p3 does not get merged
        b.add(p3);
        assertEquals(2, b.size(), "Size Error");

        //Test that p4 merges two into p3
        b.add(p4);
        assertEquals(2, b.size(), "Quantity Error");
    }

    @Test
    public void testSortAddProduct() {
        BetterBasket b = new BetterBasket();
        Product p1 = new Product("0001", "Toaster", 10.00, 1);
        Product p2 = new Product("0002", "Microwave", 50.00, 1);
        Product p3 = new Product("0003", "Gaming PC", 1000.00, 1);

        //Test that p1 and p2 get merged
        b.add(p3);
        b.add(p1);
        assertEquals("0001", b.get(0).getProductNum(), "Missorted");
        assertEquals("0003", b.get(1).getProductNum(), "Missorted");

        //Test that p2 gets inserted
        b.add(p2);
        assertEquals("0001", b.get(0).getProductNum(), "Error");
        assertEquals("0002", b.get(1).getProductNum(), "Error");
        assertEquals("0003", b.get(2).getProductNum(), "Error");
    }
}
```

With both test methods we use the assertEquals which will tell you if two objects are equal, we first check if the products merged correctly then we tested to see if they were sorted correctly.

Finished after 0.098 seconds

Runs: 2/2

Errors: 0

Failures: 0

✓ BetterBasketTest [Runner: JUnit 5] (0.001 s)

testMergeAddProduct (0.001 s)

testSortAddProduct (0.000 s)

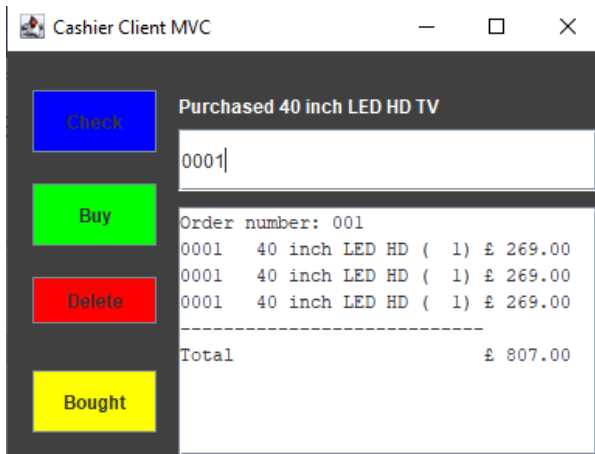


Figure 1 Before

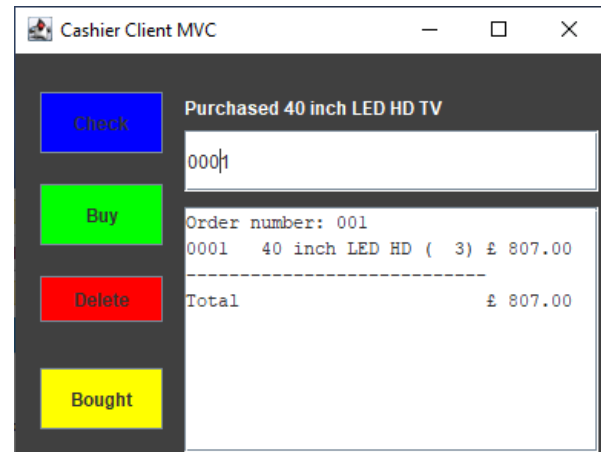


Figure 2 After products are merged

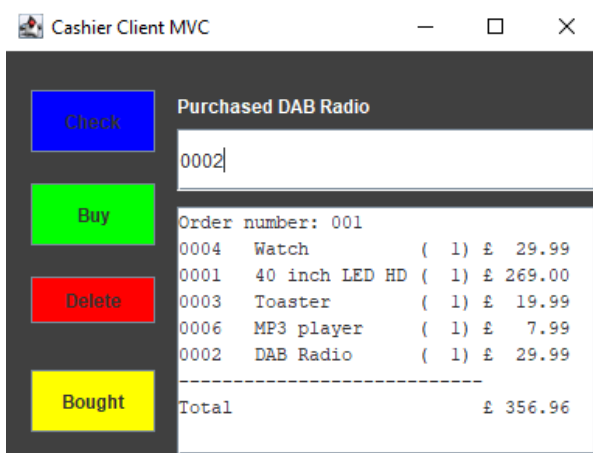


Figure 1 Before

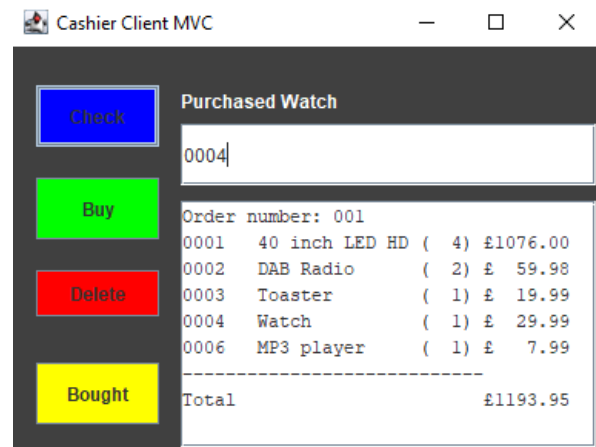


Figure 2 After products are ordered by product number

Now we are done testing our new class we can push it to our main repository and integrate it into Jenkins and create a BetterBasketBuild, Test and then Deploy.

```
C:\Users\Ahmed>cd Downloads
C:\Users\Ahmed\Downloads>java -jar jenkins.war
Running from: C:\Users\Ahmed\Downloads\jenkins.war
```

First, we install Jenkins from the command prompt, once installed the server will be running.

Next, we connect to <http://localhost:8080/> and create a new build and test models as freestyle projects, we enter our descriptions and link our GitHub repository and set the branch specifiers in this case the BetterBasket branch. As part of our build steps, we will add a call to the compile script which will be used in building/testing.

Source Code Management

☐ None

☒ Git ?

Repositories ?

Repository URL ?

<https://github.com/ae73/catshop.git>

Execute Windows batch command ?

Command

See [the list of available environment variables](#)

%WORKSPACE%\cat_compile

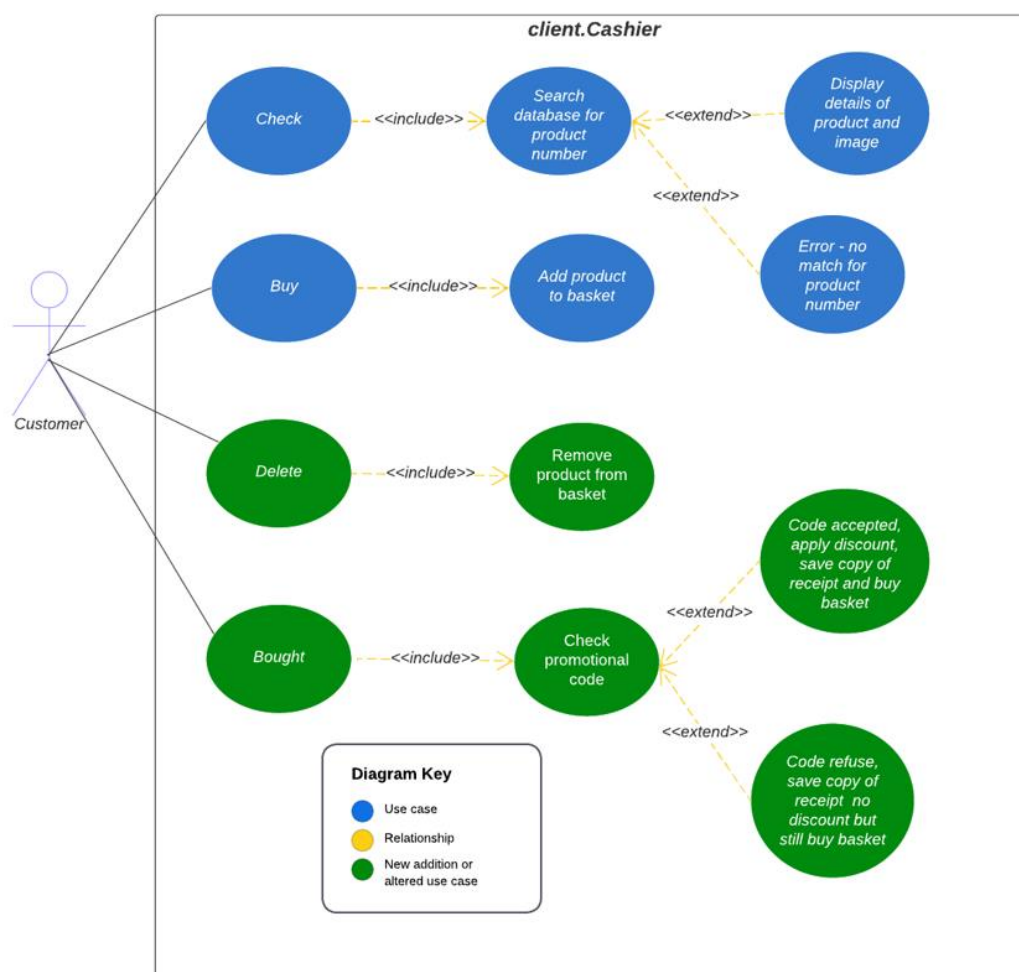
Once we have our build and test completed and successful, we can then deploy the build. Our main repository on GitHub is updated with this build and we can create a new branch and begin a new feature.

S	W	Name ↓
✓	☀	BetterBasketBuild
✓	☀	BetterBasketDeploy
✓	☀	BetterBasketTest

(What Is Jenkins? Simplilearn, 2018)

A UML case diagram of the Cashier client with my planned improvements to the class, they include:

- A remove button to remove the last item added to the basket.
- A promotional code implementation using JOptionPane that will apply a 10% discount.
- A receipt feature that will save a copy of the purchased basket to a text file.



(Nishadha. Use case diagram relationships explained with examples, 2022)

doRemove Button

In the CashierModel I created a new method called doRemove this method will remove the last item added to the basket as well as re-adding that item back to stock.

```

/*doRemove - a method to remove a particular item from the basket and to re-add that item to stock
 * it is essentially the reverse of the doBuy and doBought methods*/
public void doRemove() {
    String theAction = "";
    try
    {
        if ( theBasket.isEmpty() )           // check basket
        {
            theAction = "Your basket is empty!!..";
        }else if ( theState != State.checked ) // check |
        {
            // with customer
            theAction = "Check if OK with customer first";
        } else {
            Product prod = theBasket.remove(); // call remove on the product in basket
            theStock.addStock( prod.getProductNum(), 1); // re-add product to stock
            theAction = "Removed " + // print action
                prod.getDescription(); //
        }
    } catch( StockException e )
    {
        DEBUG.error( "%s\n%s",
            "CashierModel.doBuy", e.getMessage() );
        theAction = e.getMessage();
    }
    theState = State.process; // All Done
    setChanged(); notifyObservers(theAction);
}

```

To implement the doRemove() method I altered the doBuy method that was provided for use but reversed the increments or additions. It first checks the basket to see if it is

empty, then check with customer, then call remove on the product in the basket and re-add the product to stock with a confirmation print. Manually testing the code and checking stock was re-added.

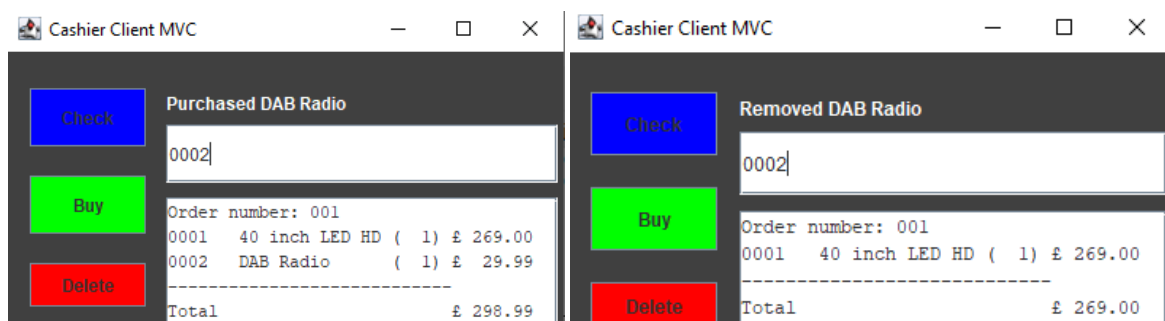


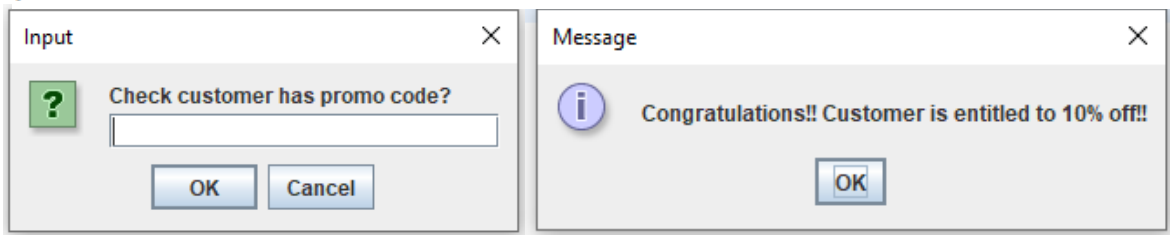
Figure 1 Before

Figure 2 After using the delete button

doBought Promotional Offer

While still in the CashierModel I can implement a promotional offer that would happen in a real-world environment – to begin with I create a new branch doPromo and in the doBought method I added a JOptionPane which will ask the employee if the customer has a promotional code, if the code provided equals the code, then another Pane will appear with confirmation if not the rest of the doBought code runs.

```
String promo= JOptionPane.showInputDialog ("Check customer has promo code?");
if(promo.equals("promo")) {
    JOptionPane.showMessageDialog(null, "Congratulations!! Customer is entitled to 10% off!!");
}else {
    JOptionPane.showMessageDialog(null, "Sorry!! Customer is NOT entitled to 10% off!!");
}
```



(Balasooriya, N. *JOptionPane*, 2012)

Now to implement the code that'll take 10% off the total, in the Basket class we first need to calculate the total cost of the basket – to calculate the total we simply loop through the ArrayList of products and for each product in the basket we get its price and multiply it by the basket quantity and keep a running total.

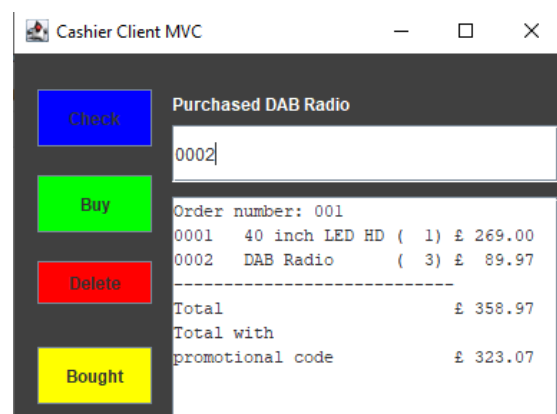
```
/*Method that loops through theBasket arraylist and multiplies each price by their quantity and keeps a running total*/
public double getTotal()
{
    double total = 0.0;
    for ( Product pr: this )
    {
        total += pr.getPrice() * pr.getQuantity();
    }
    return total;
}
```

To get the promotional discount of 10% we need to get the total cost divide it by 100 and then multiply the result by 90.

```
/*Method to generate discount, as default its 10% off with a promotional code*/
public double getPromo()
{
    return (getTotal() / 100)*90 ;
}
```

While still in the Basket class we can alter the getDetails method to output our total and total with promotional offer.

```
fr.format("Total\n", csign, getTotal());
fr.format("Total with \npromotional code\n", csign, getPromo());
fr.close();
```



doReceipt Feature

The getDetails method is very similar to a receipt, to implement this I will be saving orders to a text file and saving it in the root folder or in a real-world setting the text file could be printed and given to the customer as a receipt.


In the doBought method in CashierModel I imported all the required libraries and implemented the following code:

```
DateTimeFormatter customFormatter = DateTimeFormatter.ofPattern("dd-MM-yyyy"); //Will formate the date the way you like
LocalDateTime now = LocalDateTime.now(); //Retreives the date from local computer
//Create a new file object with a custom name which will have their order number followed by the date
//and the .txt file extension
File file = new File(theBasket.getOrderNum()+"["+customFormatter.format(now)+"]"+".txt");
file.createNewFile();
//Once file is created we need to write the details, first we create a filewriter object which takes the file as a parameter
//seems strange to use 3 different writer classes but it will not run without them
FileWriter fw = new FileWriter(file, true);
BufferedWriter bw = new BufferedWriter(fw); // the buffered writer object then takes the filewriter object as a parameter
PrintWriter pw = new PrintWriter(fw); // and then the printwriter object takes the filewriter object as a parameter too
pw.println(theBasket.getDetails()); //finally we print the basket details using the getDetails method and close
pw.close();
```

(manastole01. Java program to save a string to a file, 2021)

Using DateTimeFormatter we can give our date a custom format, this is essential in combination with the order number to create a unique text file, LocalDateTime will retrieve the date from your local machine. We then create a new file using a file object and assign its name when saving which will be the order number followed by the custom date format and with an extension of ".txt", we then go through the different writers so we can then print theBasket details to a text file.

(Gupta, L. Java datetimeformatter, 2022)

 1[10-01-2023].txt

The file is saved in the root folder with its order number followed by the date as its name.

File	Edit	Format	View	Help
Order number: 002				
0005	Digital Camera (1)	£	89.99

Total			£	89.99
Total with				
promotional code			£	80.99

The contents of the file – its order number, purchases, total and total with promotional cost.

Critical Review

Three reasons why the design, management and implementation of the project are good:

- I believe the design of the application is decent, I implemented features that would imitate a real-world store with promotional offers and receipts and multiple quality of life features for both customer and employee with the BetterBasket/BetterCashierModel, improved GUI with sound and a delete button for the employee.
- The planning and management were very good, having a simple project cycle with a combination of GitHub and Jenkins and the use of UML diagrams to visualise the interactions within a client really aided in the planning and progression of the development cycle - each new feature or alteration to the system was methodically tested using Junit tests, Jenkins test builds as well as manual testing.
- I believe the implementation of the project was also good with reasonably neat and well commented code to understand how it works, the code runs exactly the way it was intended and does not create any bugs, lag or any other issues within the application.

Three reasons where the implementation could be improved and a summary of how the improvements could be made:

- I'd like to have gotten an advertisement window with a slideshow of available products, this could have been implemented by creating a new JFrame and adding the product images to an array it could have been possible to loop through it using a timer.
- I would have also liked to have implemented JavaFX into the system, this could have been done by importing JavaFX and refactoring a lot of code implementing Application rather than Observer, changing the ContentPanels to Stages etc. I'd also have been able to use mp3 files for sound rather than wav, Java Swing seems to struggle with audio even wav files over a certain bandwidth will not be playable.
- Completely removing the Observer pattern and replacing it with EventListeners, would have involved refactoring a lot of code but EventListeners are more secure and not deprecated like Observer.

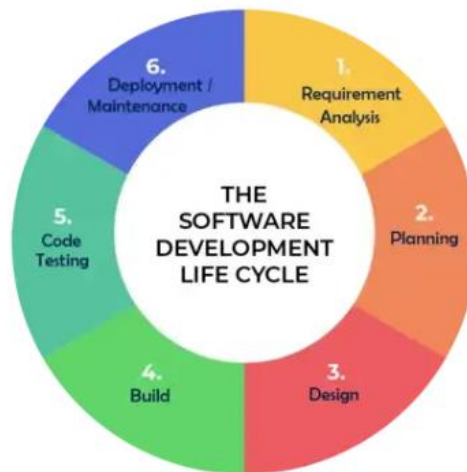
Conclusion

What are the key concepts that you learned during the development:

One of the key concepts I took away was project planning – comparing this project to my first-year project they are worlds apart, having a well-structured project plan kept new features being integrated smoothly. I felt it improved my own efficiency and made it easy to track with well-defined project goals.

Another key concept was the development cycle and integrating GitHub and Jenkins, it's an ideal way to design and implement new software or applications with its continuous integration – you can perfectly follow the development cycle of planning, design, build, test then deploy.

UML case diagrams were also very useful when trying to breakdown an application even more so if it were a larger application with many more classes and potentially packages, it was also great for keeping track of what has been updated and what needs to be updated.



(Karunajeewa, S. *Software development life cycle*, 2020)

Estimated Grade:

Development: A, Management: B, Report: A

Development: 65-75%, Management: 60-70%, Report: 65-75%

References

- Balasooriya, N. (2012) *Joptionpane and getting text from input*, *Stack Overflow*. Available at: <https://stackoverflow.com/questions/8346650/joptionpane-and-getting-text-from-input>.
- Gupta, L. (2022) *Java datetimeformatter with examples*, *HowToDoInJava*. Available at: <https://howtodoinjava.com/java/date-time/java8-datetimeformatter-example/>.
- Karunajeewa, S. (2020) *Software development life cycle methods, their advantages and disadvantages*, *Medium*. Bootcamp. Available at: <https://bootcamp.uxdesign.cc/sdlc-methods-and-their-advantages-and-disadvantages-80095cea796c>.
- manastole01 (2021) *Java program to save a string to a file*, *GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/java-program-to-save-a-string-to-a-file/>.
- Nishadha (2022) *Use case diagram relationships explained with examples*, *Creately Blog*. Available at: <https://creately.com/blog/diagrams/use-case-diagram-relationships/>.
- Rosencrantz, N. (2020) *Playing sound in java*, *Stack Overflow*. Available at: <https://stackoverflow.com/questions/25171205/playing-sound-in-java>.
- What Is Jenkins? | What Is Jenkins And How It Works? | Jenkins Tutorial For Beginners | Simplilearn* (2018) *Simplilearn YouTube Channel*. YouTube. Available at: <https://www.youtube.com/watch?v=LFDnKPOTg>.